Advancements in Symbolic Traversal Techniques *

Gianpiero Cabodi and Paolo Camurati

Politecnico di Torino, Dipartimento di Automatica e Informatica, Turin, Italy

Abstract. Symbolic state space traversal techniques are one of the most notable achievements in the fields of formal verification and of automated synthesis. Transition functions and transition relations are two alternative approaches. In terms of efficiency, transition functions have proven to be superior, although the transition relation is much more expressive. This paper brings the transition relation back to a new life, profiting from recent advancements in the fields of boolean function representation, simplification, and image computation represented by BDDs and by the generalized cofactor operator. A theoretical result allows us to considerably simplify both the process of building the transition relation and of traversing the state space. Experimental results show that performances similar to those of the transition function are obtained.

1 Introduction

Dealing with Finite State Machines (FSMs) [15] in the fields of formal verification of correctness, of automated synthesis, and of testing essentially requires a traversal of their state space, where properties, such as equivalence, language containment, or temporal formulæ are checked.

There are three major problems to be solved to reach the goal: an efficient representation for boolean functions, an efficient way to represent and manipulate sets, namely sets of states, and good algorithms for exploring the space state. *Binary Decision Diagrams* (BDDs) [6], [5] are currently the most popular answer to the first issue. *Characteristic functions* are used to represent sets and, being boolean, they can benefit from the efficiency of BDDs. Symbolic traversal techniques [7], [8], [3], [9] complete the framework, supporting reachability analysis of real-size FSMs.

Analyzing reachable states essentially requires to represent the state transition function of the FSM and then to traverse it, either backwards or forwards. In forward traversal, starting from a set of initial states, the next ones are computed, i.e., the image of the state transition function on that set is evaluated. In backward traversal, starting from a set of states, specified according to the application, the ones from which that set is reached in one step are computed, i.e., a pre-image of the state transition function is evaluated.

* This work has been partially supported by the ESPRIT Working Group 6018 "CHARME-2".

There are two approaches to the representation of the state transition function, each with its own algorithm for image computation:

- the "*transition relation*";
- the "*transition function*".

With the "*transition relation*" [7], [3] the sequential behaviour of the FSM is described by listing the couples "current state - next state", independently of the inputs. Such a list is represented by its characteristic function, which in turn is a BDD. Computing the image of the state transition function on a given set of states is quite easy, as it is enough to consider only those couples "current state - next state" of the transition relation where the current state belongs to the domain being considered and to return the corresponding next states. These operations are easily performed on BDDs, as they consist in and-ing two BDDs, existentially quantifying ("*smoothing*") the current states, and then relabeling the resulting next states as the new current ones [3]. The FSM can easily be traversed forwards or backwards, as computing images or pre-images makes no difference. The main limit resides in the difficulty to build the transition relation for big circuits, i.e., to first and many boolean functions and then to existentially quantify the inputs. As existential quantification does not distribute over and-ing, the resulting BDD rapidly grows too large.

In order to overcome this limit, the approach based on the "*transition function*" [8], [12], [13] was developed, i.e., on a vector of boolean functions, each representing the behavior of the FSM along one of the dimensions in the space. The recursive image computation algorithm for transition functions exploits the powerful operator "*cofactor*" [8], [11], that allows one to simplify a function when it is constrained on a subset of its domain. Intuitively, this means that the resulting function coincides with the original one on the subset of the domain defined by the constraint, whereas for the rest of the domain its value can be chosen arbitrarily. Among the many possible choices, an appropriate one allows to guarantee that the image of the original function on the constrained domain and of the simplified function on the whole domain are the same ("*generalized cofactor and image restrictor*"). In the image computation algorithm, the state transition function is first simplified on the subset of its domain currently being considered, then a recursive procedure is applied, based on the expansion theorem, that computes the result by either splitting the inputs or the outputs [11]. This second approach, although the image computation algorithm is not as simple as the one for the transition relation, allowed a major breakthrough in terms of efficiency and applicability. The transition function is less general than the transition relation, as the former must be deterministic, the latter, though implemented by BDDs, easily supports non-determinism, used for example with Process Algebras [3], [14], [10]. Moreover, whereas pre-image computation is immediate for the transition relation, it is more awkward for transition functions.

This brief discussion justifies the fact that neither approach has been dismissed in favor of the other and that research on both is still ongoing. In order to be able to build the transition relation, partitioning techniques are introduced, but either they are manual and based on the designer's knowledge [2] or

they may be computationally expensive [4]. This paper follows another avenue of attack to bring the transition relation back to a new life. Existential quantification and and-ing do not distribute, but, as one of the effects of the generalized cofactor and image restrictor is to propagate the constrain while simplifying the functions, it is possible to first apply it and then to distribute existential quantification and and-ing. A minor condition is imposed on the way the cofactor is built, preserving all its usual properties. This makes building the transition relation feasible. Experimental results show that the CPU time is reduced by an order of magnitude with respect to the basic method and it is comparable with the best results obtained with the transition function approach. The same considerations hold for the size of the BDDs.

Section 2 shows how the generalized cofactor and image restrictor can successfully be used to build the transition relation. Section 3 presents in more detail how the theoretical results are applied to reachability analysis. The experimental data on the ISCAS'89 [1] circuits of Section 4 support the claim that the transition relation is again competitive with the transition function.

2 Theoretical Framework

A completely specified FSM M is 6-tuple $M = (I, O, S, \delta, \lambda, S_0)$, where I (O) is the input (output) space, S is the state space, δ (λ) is the state transition (output) function, and S_0 is the set of initial states. Without any loss of generality, we hereinafter consider boolean spaces only, as other spaces can be mapped into them by suitable binary encodings.

When working on boolean spaces, denoting with B the set $\{0, 1\}$, I, O, S_0, and S are powers of B and δ and λ are functions from powers of B to powers of B. In particular, let us suppose $I = B^m$, m being the number of inputs, $O = B^k$, k being the number of outputs, $S = B^n$, n being the number of state variables, then, in the general case of Mealy machines, $\lambda : B^m \times B^n \rightarrow B^k$ and $\delta : B^m \times B^n \rightarrow B^n$.

Let A be a subset of B^n: its "*characteristic function*" $\chi_A : B^n \rightarrow B$ is defined as follows:

$$\chi_A(a) = \begin{cases} 1 \text{ if } a \in A \\ 0 \text{ otherwise} \end{cases}$$

Set operations are efficiently implemented by boolean operators on BDDs.

The state transition function δ can be represented either as a transition function [8] or as a transition relation [7], [3]. Let us follow the second choice.

The characteristic function $\delta_c : B^m \times B^n \times B^n \rightarrow B$ corresponding to δ returns 1 iff the next state $y \in S$ is the image of the current state $s \in S$ and of the input $x \in I$ according to function δ. As we are often interested only in the existence of an input value, rather than in the value itself, the "*transition relation*" abstracts from the inputs. We write, with abuse of notation, T_M, instead of χ_{T_M}, for the characteristic function of the transition relation, defined as:

$$T_M(s, y) = \exists x \prod_{i=1}^{n} (y_i \equiv \delta_i(x, s))$$

Once the transition relation is given, computing the image (pre-image) of δ on a set of states described by its characteristic function $C(s)$ ($C(y)$) is relatively easy, as it is enough to and it with the transition relation, to existentially quantify, and to relabel the states:

$$\text{IMG}(\delta, C(s)) = \exists s T_M(s, y) \cdot C(s)_{y/s}$$

$$\text{PREIMG}(\delta, C(y)) = \exists y T_M(s, y) \cdot C(y)_{s/y}$$

The transition relation T_M is conceptually easy to build, but often, for medium size and large circuits, it requires too much time and too many BDD nodes. Individual terms in the product of the form $(y_i \equiv \delta_i(x, s))$ have relatively small BDDs, but existential quantification and logical and do not distribute, unless the sets of support variables are disjoint. As a consequence, it is necessary to build the whole product $\prod_{i=1}^{n}(y_i \equiv \delta_i(x, s))$ before applying the existential quantifier and this is often impossible, as the resulting BDD is too large.

Finding disjoint sets of support is one possible avenue of attack, though literature reports only manual methods [2] or expensive ones [4]. Our approach exploits the generalized cofactor and image restrictor to distribute existential quantification and logical and, so that the BDDs stay reasonable in size.

Let us first formally define the generalized cofactor and image restrictor $\downarrow$. For boolean functions $f(x)$ and $g(x)$, given an ordering for the xs, $f \downarrow g$, or f constrained by g, is defined by [8], [11]:

$$(f \downarrow g)(x) = \begin{cases} f(x) & \text{if } g(x) = 1 \\ f(x_0) & \text{if } g(x) = 0 \end{cases}$$

where x_0 is the minterm such that $g(x_0) = 1$ and the distance $||x - x_0||$ is minimal.

The operator satisfies the following identity:

$$g \cdot f = g \cdot (f \downarrow g)$$

The following lemma and theorem form the basis for distributing existential quantification and logical and by means of a suitable definition of the generalized cofactor and image restrictor. The lemma is a particular case of the general result proven in [16].

Lemma 1. *Let us consider a function f of the form $f(x) = \prod_{i=1}^{n} f_i(x)$ and a constraint $c(x)$, then $f(x) \downarrow c(x) = \prod_{i=1}^{n}(f_i \downarrow c)(x)$.*

Proof. As $f(x) = \prod_{i=1}^{n} f_i(x) = f_1(x) \cdot \prod_{i=2}^{n} f_i(x) = a(x) \cdot b(x)$, we can prove the thesis by restricting investigation to the 2 term case. The thesis is rewritten as:

$$(f \downarrow c)(x) = (a \downarrow c)(x) \cdot (b \downarrow c)(x)$$

To prove the thesis, we split the cases according to the value of $c(x)$:

- $c(x) = 1$:

$$(f \downarrow c)(x) = f(x) = (a \downarrow c)(x) \cdot (b \downarrow c)(x) = a(x) \cdot b(x)$$

- $c(x) = 0$: let us remind that the choice of x_0 depends only on the constraint $c(x)$ and on variable ordering. Selecting the same x_0

$$(f \downarrow c)(x) = f(x_0) = (a \downarrow c)(x) \cdot (b \downarrow c)(x) = a(x_0) \cdot b(x_0)$$

□

Theorem 2. *Hypothesis: let us define the* $\downarrow$ *operator between two functions* $a(x, y)$ *and* $b(x, y)$ *as follows:*

$$(b \downarrow a)(x, y) = \begin{cases} \textit{if } a(x, y) = 1 \textit{ then } b(x, y) \\ \textit{if } a(x, y) = 0 \textit{ then if } \exists x_0\, a(x_0, y) = 1 \textit{ then} \\ \qquad\qquad\qquad\qquad\qquad\qquad b(x_0, y) \\ \qquad\qquad\qquad\qquad\qquad\qquad \textit{else} \\ \qquad\qquad\qquad\qquad\qquad\qquad b(x_0, y_0) \,|\, a(x_0, y_0) = \end{cases}$$

There is no change in the way the cofactor is computed, provided that variable ordering puts the ys before the xs.

Thesis:

$$\exists x\, a(x, y) \cdot b(x, y) = \exists x\, a(x, y) \cdot \exists x\, (b \downarrow a)(x, y)$$

Proof. By applying the rewrite rule $f = g \cdot f + \bar{g} \cdot f$, $\exists x\, a(x, y) \cdot \exists x\, (b \downarrow a)(x, y)$ becomes:

$$\exists x\, a(x, y) \cdot \exists x\, (a(x, y) \cdot (b \downarrow a)(x, y) + \bar{a}(x, y) \cdot (b \downarrow a)(x, y))$$

and, by applying $g \cdot (f \downarrow g) = g \cdot f$, we obtain:

$$\exists x\, a(x, y) \cdot \exists x\, (a(x, y) \cdot b(x, y) + \bar{a}(x, y) \cdot (b \downarrow a)(x, y)) =$$

$$\exists x\, a(x, y) \cdot \exists x\, a(x, y) \cdot b(x, y) + \exists x\, a(x, y) \cdot \exists x\, (\bar{a}(x, y) \cdot (b \downarrow a)(x, y))$$

As $\exists x\, a(x, y) \cdot b(x, y) \Rightarrow \exists x\, a(x, y)$, the previous formula becomes:

$$\exists x\, a(x, y) \cdot b(x, y) + \exists x\, a(x, y) \cdot \exists x(\bar{a}(x, y) \cdot (b \downarrow a)(x, y))$$

and, applying the definition of cofactor to $\exists x(\bar{a}(x, y) \cdot (b \downarrow a)(x, y))$, taking into account that the term is logically and-ed with $\exists x\, a(x, y)$, the previous formula becomes:

$$\exists x\, a(x, y) \cdot b(x, y) + \exists x\, a(x, y) \cdot \exists x(\bar{a}(x, y) \cdot b(x_0, y) \cdot a(x_0, y))$$

This equals $\exists x\, a(x, y) \cdot b(x, y)$, as whenever the left-hand side is true, the right-hand side is also true and $\exists x\, a(x, y) \cdot b(x, y) = 0 \Rightarrow b(x_0, y) \cdot a(x_0, z) = 0$. □

The theorem holds also in the general case of the form $f(x, y) = \prod_{i=1}^{n} f_i(x, y)$, as $f(x, y) = \prod_{i=1}^{n} f_i(x, y) = f_1(x, y) \cdot \prod_{i=2}^{n} f_i(x, y) = a(x, y) \cdot b(x, y)$.

3 An Application: Reachability Analysis

Computing the reachable state space finds applications in equivalence proofs for FSMs, in ATPG, and in the manipulation of Process Algebra formulæ. In symbolic breadth-first traversal multiple states, represented by their characteristic function, are considered simultaneously, evaluating function δ on them, the result being the characteristic function of the set of states reachable from them in one step. The pseudo-code is shown in Fig. 1.

```
var Reached, From, New, Image: set of S;
begin
    Reached := From := S0 ;
    repeat
        Image := IMG(δ, From) ;
        New := Image ∩ (Reached) ;
        Reached := Reached ∪ Image ;
        From := New ;
    until New = ∅ ;
    return Reached;
end
```

Fig. 1. Symbolic Breadth-First Traversal

Computing the image requires the knowledge of the transition relation T_M, whose construction is often impossible, unless the theorem proven in the previous Section is used. The experimental results of the next Section confirm this statement. The theorem can be applied by building a priori the transition relation (algorithm A) or building it in a constrained and simpler form at each traversal step (algorithm B).

Algorithm A The theorem of the previous Section helps building once and for all the transition relation T_M:

$$T_M(s, y) = \exists x \left(\prod_{i=1}^{n} (y_i \equiv \delta_i(x, s))\right) = \exists x \left(\prod_{i=1}^{n} f_i(x, s, y)\right)$$

Considering each term f_i and omitting the argument variables for simpler notation, let us write:

$$f_i^{(1)} = f_i \qquad 1 \leq i \leq n$$

$$f_i^{(j)} = f_i^{(j-1)} \downarrow f_{j-1}^{(j-1)} \qquad 2 \leq j \leq n \text{ and } j \leq i \leq n$$

i.e., let us iteratively simplify each term by cofactoring it with previously simplified ones. The superscript indicated the simplification step. With abuse of notation, let us write $f_k^{(k)} = f^{(k)}$. The transition relation is then recursively expressed as:

$$T_M = T_M^{(1)}$$

with

$$T_M^{(k)} = \exists x \, f^{(k)} \cdot T_M^{(k+1)}$$

$$T_M^{(n)} = \exists x \, f^{(n)}$$

Applying the cofactor propagates the constraints inside the functions and allows to reduce their complexity by simplification and early quantification.

Computing IMG(δ, From) is easy once T_M is given. Set From is described by its characteristic function $C(s)$. Extending it so that it ranges over variables x, s, y, such a function can be considered as the $n+1$-th term of the product, i.e., $C(x, s, y) = f_{n+1}(x, s, y)$. The traditional approach for image computation is expressed by the following formula:

$$\text{IMG}(\delta, \text{From}) = \exists s \exists x \left(\prod_{i=1}^{n+1} f_i(x, s, y)\right) \tag{1}$$

Applying the theorem yields:

$$\text{IMG}(\delta, \text{From}) = \prod_{i=1}^{n+1} (\exists s \exists x \, (f^{(i)}(x, s, y))) \tag{2}$$

and the advantages are evident, as an early quantification takes place on the simplified $f^{(i)}$ terms. Moreover, the first n terms of the product can be computed once and for all.

Algorithm B Instead of building T_M and then computing the image, if the goal is merely forward state space traversal, it is possible to build at each step the transition relation, as in algorithm A, conveniently simplified on the set of starting states From.

One can easily demonstrate that in the traditional approach the following formula holds:

$$\text{IMG}(\delta, \text{From}) = \text{IMG}(\delta \downarrow C(s), 1) = \exists s \exists x \left(\prod_{i=1}^{n}((f_i \downarrow C)(x, s, y))\right) \quad (3)$$

Applying the theorem yields, as easily demonstrated:

$$\text{IMG}(\delta, \text{From}) = \text{IMG}(\delta \downarrow C(s), 1) = \prod_{i=1}^{n}(\exists s \exists x((f \downarrow C)^{(i)}(x, s, y))) \quad (4)$$

with a considerable reduction in the number of BDD nodes and in CPU time.

4 Experimental Results

All experiments have been performed by a fully home made software on a 30 Mips VAX-9000. The number of BDD nodes was limited to 2,000,000, and garbage collection was disabled (except for a few cases) to allow exact quantification of the generated BDDs .

Tab. 1 and 2 show the experimental data collected on the ISCAS'89 circuits [1] that are currently dealt with by symbolic techniques. The column labelled TF refers to the results obtained with the transition function and published in [9]. The column labelled TR-A refers to the traditional approach of first building once and for all the transition relation and then traversing the FSM (equation (1)). The approach whose results are shown in column TR-B differs from the former in the preliminary simplification of the state transition function δ on the initial state set From at each new image computation, before the transition relation is built (equation (3)). Columns labeled Alg-A and Alg-B show the data obtained by applying equations (2) and (4).

The last column, labeled Alg-Bsort, refers to a heuristic method, oriented to obtain mimimum size BDDs for state sets. This is achieved through a dynamic ordering for the variables, keeping on top of the BDDs the variables that are most discriminating. The δ_is are sorted from left to right in increasing complexity, assuming that the more complex the δ_i function, the more discriminating the corresponding y_i. We measure BDD complexity by counting the number of nodes of its tree expansion (BDD without sharings). The overhead incurred by this ordering algorithm is due to the need to either reconvert the δ_is in the next state variables' space or the next states to the space of the δ_is. Good results are obtained with some of the circuits involving larger BDDs.

Circuit	# states	BDD nodes					
		TF	TR-A	TR-B	Alg-A	Alg-B	Alg-Bsort
s298	218	874	52777	5037	4185	1229	2004
s344	2625	11233	230247	62422	23363	19058	35319
s349	2625	11233	230247	62422	23363	19058	35319
s382	8865	19746	375901	319535	73065	33092	66227
s386	13	255	575	581	519	362	404
s400	8865	19746	375901	319535	73065	33092	66227
s420	17	554	11980	1372	2003	421	391
s444	8865	8066	730998	196768	62654	20704	44149
s510	47	469	1036	1229	1615	527	522
s526	8868	9285	- *	149982	90099	22060	53534
s526n	8868	9285	- *	149982	90099	22060	53534
s641	1544	21511	590937*	234298	76533	34154	30530
s641n	1544	22001	490942*	213723	66498	30450	30861
s713	1544	21585	590941*	234323	77832	35097	30659
s820	25	432	1005	1005	1015	694	713
s832	25	432	1005	1005	1015	694	713
s838	17	1310	- *	3852	8775	922	899
s953	504	9722	44993	76387	13443	17158	22968
s1196	2616	53158	239397	322783	49281	89003	74093
s1238	2616	155375	497813	919470	57082	205186	217258
s1488	48	636	1445	1609	1747	1026	1161
s1494	48	635	1454	1618	1735	1022	1154

Table 1. Experimental results on BDD nodes: Number of BDD nodes $\leq 2,000,000$. - means overflow on BDD nodes, * means garbage collection active.

It is easy to conclude that applying equations (2) and (4) improves by an order of magnitude on the traditional transition relation and has performances comparable with the transition functions. We thus conclude that the transition relation is again competitive in terms of performances with the transition function, although it is much more powerful in terms of expressiveness.

Indeed, experimental data on the state set generation for the ISCAS'89 circuits show that our method has a complexity similar to the transition function, as the increase in the number of nodes is mainly due to variable relabeling.

From a more theoretical point of view, for the transition function approach the number of recursive IMG calls is roughly proportional to the number of nodes of the image BDD. For each recursive call, an array of functions (derived from the δ_is) is considered and all the functions, starting from the second one, are cofactored with the first one or with its complement. The so-called sharing or sub-tree recombination effect can strongly modify this proportionality, because equal BDDs are computed many times as partial images in different sub-spaces, although they are shared in the resulting BDD. Two important techniques have been introduced to avoid multiple evaluations: hash-based image evaluation [11], and disjoint set partitioning [13]. Good results have been obtained, but both

Circuit	CPU time [s]					
	TF	TR-A	TR-B	Alg-A	Alg-B	Alg-Bsort
s298	0.1	10.6	0.3	0.6	0.2	0.2
s344	1.1	34.2	5.8	2.4	1.8	2.5
s349	1.1	35.2	5.7	2.5	2.0	2.8
s382	2.3	1797.1	30.5	9.3	5.1	6.5
s386	0.1	0.1	0.1	0.1	0.1	0.1
s400	2.2	1779.7	31.0	9.5	5.1	6.4
s420	0.0	2.2	0.1	0.3	0.1	0.1
s444	1.8	794.6	17.5	7.1	3.3	4.4
s510	0.1	0.1	0.1	0.2	0.1	0.1
s526	1.7	-	11.3	10.6	3.1	4.9
s526n	1.6	-	11.2	10.7	3.2	5.0
s641	2.4	405.3	37.6	10.9	3.4	2.9
s641n	2.7	482.6	38.6	10.1	3.3	2.9
s713	2.3	404.2	38.1	11.0	3.7	2.9
s820	0.1	0.2	0.1	0.1	0.1	0.1
s832	0.0	0.2	0.1	0.1	0.1	0.1
s838	0.1	-	0.3	1.2	0.1	0.1
s953	0.9	6.6	5.1	2.2	1.7	1.8
s1196	9.1	74.4	101.7	45.3	14.3	8.1
s1238	25.8	97.3	481.1	49.9	34.3	27.7
s1488	0.0	0.2	0.3	0.2	0.1	0.1
s1494	0.0	0.2	0.3	0.2	0.1	0.1

Table 2. Experimental results on CPU time: - means overflow on BDD nodes.

level	# states	TF		TR-B	
		BDD nodes	CPU time [s]	BDD nodes	CPU time [s]
7	33698553	578170	213	414818	169
8	111100409	1110781	991	471322	542
9	489606397	-	-	832653	2450
10	1682875721	-	-	1523625	9045

Table 3. s1423: the first unmanageable ISCAS'89: garbage collection is active, *level* is the number of clock cycles required to reach all the states in the state set.

methods introduce considerable computational overheads.

For the method we propose the number of recursive calls is n, where n is the number of state transition functions and it is reduced to a linear iteration. Each iteration step is quite complex, because on top of the cofactored δs we build the BDDs of the next state variables. Handling sharings and subtree recombinations, on the other hand, results automatically: at each iteration the terms have all possible local sharings, quantifications are performed individually, and thus sharings are kept as long and as much as possible. This effect is not readily apparent in the ISCAS'89 circuits, as already demonstrated in [11], but we are

currently working on other cases where it results in considerable improvements. As a preliminary result in this direction, we include (Tab. 3) a profile of partial state space generation for the ISCAS'89 circuit s1423. No published data exist at present about the full state space of this circuit (74 state elements). Our experiments show a drastic advantage of the new transition relation algorithm, compared with the transition function approach and this is mostly due to the sharings possible in our new method.

5 Conclusions

Symbolic state space traversal techniques are one of the most notable achievements of the past few years that contributed to making formal verification quit research labs and enter the world of industrial applications. Automated synthesis and testing have also profited. Their limit resides in the inability to deal with other than simple FSMs, where the control part dominates over the data path. The struggle in terms of performances between the approaches based on the transition function and on the transition relation ended with the victory of the former ones. The application domain was restricted to FSMs and for this reason the expressive advantages of the transition relation were often overlooked.

The borders for hardware description and verification are rapidly moving towards higher abstraction levels, where systems are considered in terms of the events they exchange with the outside environment and are modeled by possibly non-deterministic labeled transition systems. In these cases, the transition relation has proven to be an excellent means to mechanize Process Algebra manipulations [14], [10] and the proof of Temporal Logic properties [3]. For these reasons, it is extremely important to have efficient ways to build it and to use it.

The main contribution of this paper was to bring the transition relation back to a new life, profiting from the recent advancements in the fields of boolean function representation, simplification, and image computation represented by BDDs and the cofactor operator. A theoretical result allows us to considerably simplify both the process of building the transition relation and of traversing its state space. It consists in distributing existential quantification and logical and by propagating the constraints by means of successive cofactorings. Experimental results show that performances similar to those of the transition function are obtained.

Future work will consist in the investigation of new variable orderings and of subtree recombination.

References

1. Brglez, F. Bryan, D., Koźmiński, K.: Combinatorial Profiles of Sequential Benchmark Circuits. ISCAS'89: IEEE Int'l Symposium on Circuits and Systems, Portland, OR (USA), May 1989, pp. 1929–1934

2. Burch, J.R., Clarke, E.M., Long, D.E.: Representing Circuits More Efficiently in Symbolic Model Checking. DAC'91: 28th ACM/IEEE Design Automation Conference, San Francisco, CA (USA), June 1991, pp. 403–407
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic Model Checking: 10^{20} States and Beyond. LICS'90: 5th Annual IEEE Symposium on Logic in Computer Science, June 1990, pp. 428–439
4. Bochmann, D., Dreisig, F., Steinbach, B.: A new Decomposition Method for Multilevel Circuit Design. EDAC'91: IEEE European Conference on Design Automation, Amsterdam (The Netherlands), February 1991, pp. 374–377
5. Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient Implementation of a BDD Package. DAC'90: 27th ACM/IEEE Design Automation Conference, Orlando, FL (USA), June 1990, pp. 40–45
6. Bryant, R.E.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computer, Vol. C-35, No. 8, August 1986, pp. 677–691
7. Coudert, O., Berthet, C., Madre, J.C.: Verification of Sequential Machines Based on Symbolic Execution. "Automatic verification methods for finite state systems,", J. Sifakis editor, Lecture Notes in Computer Science 407, Springer Verlag, Berlin (Germany), 1989, pp. 365–373
8. Coudert, O., Berthet, C., Madre, J.C.: Verification of Sequential Machines Using Boolean Function Vectors. IFIP Int'l Workshop on "Applied Formal Methods for Correct VLSI Design", Leuven (Belgium), November 1989, Vol. 1, pp. 111–128
9. Cabodi, G.P., Camurati, P., Corno, F., Gai, S., Prinetto, P., Sonza Reorda, M.: A new Model for Improving Symbolic Product Machine Traversal. DAC-29: 29th ACM/IEEE Desing Automation Conference, Anaheim, CA (USA), June 1992, pp. 614–619
10. Camurati, P., Corno, F., Prinetto, P.: Exploiting symbolic traversal techniques for efficient Process Algebra Manipulation. CHDL'93: IFIP Conference on Hardware Description Languages and their Applications, Ottawa (Canada), April 1993
11. Cho, H., Hachtel, G., Jeong, S.W., Plessier, B., Schwarz, E., Somenzi, F.: ATPG Aspects of FSM Verification. ICCAD-90: IEEE Int'l Conference on Computer Aided Design, Santa Clara, CA (USA), November 1990, pp. 134–137
12. Coudert, O., Madre, J.C.: A Unified Framework for the Formal Verification of Sequential Circuits. ICCAD-90: IEEE Int'l Conf. on Computer Aided Design, Santa Clara, CA (USA), November 1990, pp. 126–129
13. Coudert, O., Madre, J.C.: Symbolic Computation of the Valid States of the Sequential Machine: Algorithms and Discussion. 1991 Int'l Workshop on Formal Methods in VLSI Design, Miami, FL (USA), January 1991
14. Enders, R., Filkorn, T., Taubner, D.: Generating BDDs for Symbolic Model Checking in CCS. CAV'91: Computer-Aided Verification Workshop, Aalborg (Denmark), July 1991, K.G. Larsen, A. Skou Editors, Lecture Notes in Computer Science 575, Springer Verlag, Berlin (Germany), pp. 203–213
15. Kohavi, Z.: Switching and Finite Automata Theory. second edition, Computer Science Series, Mc Graw Hill, New York, NY
16. Touati, H., Savoj, H., Lin, B., Brayton, R.K., Sangiovanni-Vincentelli, A.: Implicit Enumeration of Finite State Machines Using BDDs. ICCAD-90: IEEE International Conference on Computer Aided Design, Santa Clara, CA (USA), November 1990, pp. 130–133